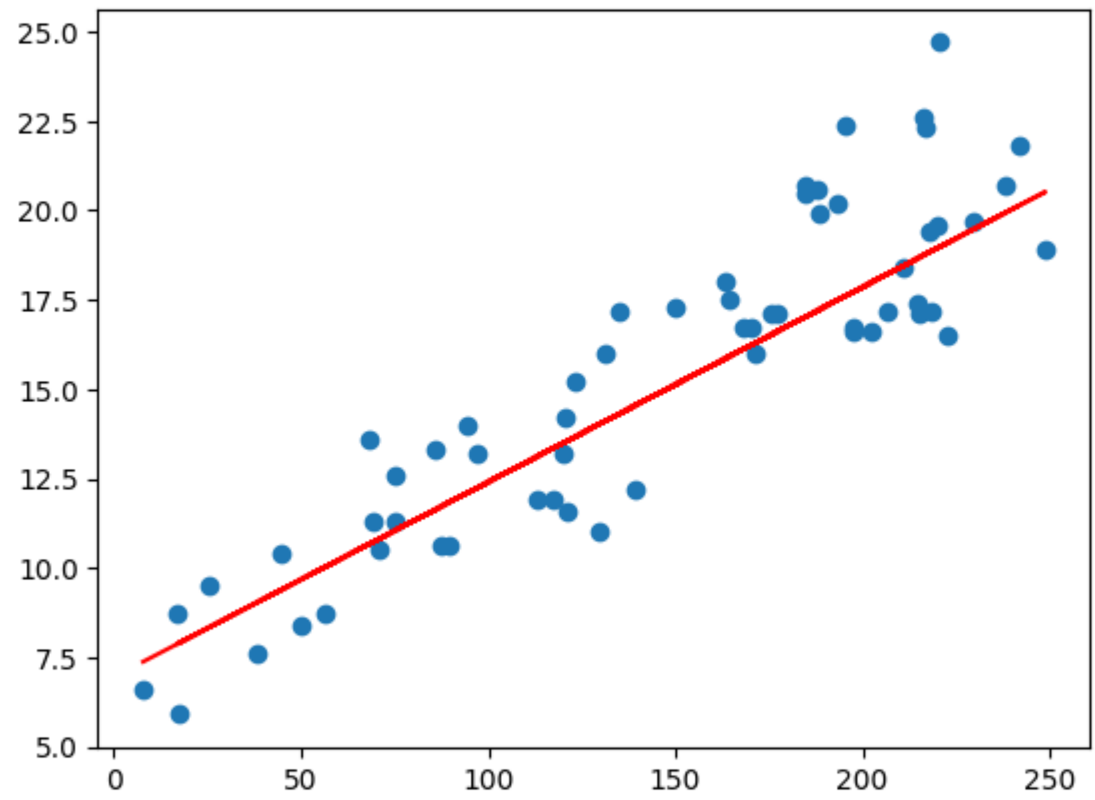
In [3]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

#Reading the dataset
dataset = pd.read_csv("tv.csv")

#Setting the value for X and Y
x = dataset[['TV']]
y = dataset['Sales']

#Splitting the dataset
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size =

#Fitting the Linear Regression model
from sklearn.linear_model import LinearRegression
slr = LinearRegression()
slr.fit(x_train, y_train)

#Intercept and Coefficient
print("Intercept: ", slr.intercept_)
print("Coefficient: ", slr.coef_)


## Regression Equation: Sales = 6.948 + 0.054 * TV

#Prediction of test set
y_pred_slr= slr.predict(x_test)
#Predicted values
print("Prediction for test set: {}".format(y_pred_slr))

#Actual value and the predicted value
slr_diff = pd.DataFrame({'Actual value': y_test, 'Predicted value': y
slr_diff.head()

#Line of best fit
plt.scatter(x_test,y_test)
plt.plot(x_test, y_pred_slr, 'Red')
plt.show()

#Model Evaluation
from sklearn import metrics
meanAbErr = metrics.mean_absolute_error(y_test, y_pred_slr)
meanSqErr = metrics.mean_squared_error(y_test, y_pred_slr)
rootMeanSqErr = np.sqrt(metrics.mean_squared_error(y_test, y_pred_sl
print('R squared: {:.2f}'.format(slr.score(x,y)*100))
print('Mean Absolute Error:', meanAbErr)
print('Mean Square Error:', meanSqErr)
print('Root Mean Square Error:', rootMeanSqErr)
```

```
Intercept:  6.948683200001357
Coefficient:  [0.05454575]
Prediction for test set: [ 7.37414007 19.94148154 14.32326899 18.82
```

```
329361 20.13239168 18.2287449
 14.54145201 17.72692398 18.75238413 18.77420243 13.34144544 19.466
93349
 10.01415451 17.1923756  11.70507285 12.08689312 15.11418241 16.232
37035
 15.8669138  13.1068987  18.65965635 14.00690363 17.60692332 16.603
28147
 17.03419291 18.96511257 18.93783969 11.05597839 17.03419291 13.663
26538
 10.6796127  10.71234015 13.5487193  17.22510305  9.67597085 13.521
44643
 12.25053038 16.13418799 19.07965865 17.48692266 18.69783838 16.532
37199
 15.92145955 18.86693021 13.5050827  11.84143724  7.87050642 20.519
66653
 10.79961336  9.03233096 17.99419817 16.29237067 11.04506924 14.099
63141
 18.44147334  9.3759692   7.88687015  8.34505447 17.72692398 11.623
254221
```




```
R squared: 81.10
Mean Absolute Error: 1.6480589869746525
Mean Square Error: 4.077556371826948
Root Mean Square Error: 2.019296008966231
```

In [8]:
```
!pip install seaborn
```

```
Defaulting to user installation because normal site-packages is not
writeable
Collecting seaborn
  Downloading seaborn-0.12.2-py3-none-any.whl (293 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 293.3/293.3 kB 2.8 MB/
s eta 0:00:00[31m2.4 MB/s eta 0:00:01
Requirement already satisfied: matplotlib!=3.6.1,>=3.1 in ./.local/
lib/python3.10/site-packages (from seaborn) (3.7.1)
Requirement already satisfied: pandas>=0.25 in ./.local/lib/python
3.10/site-packages (from seaborn) (2.0.1)
Requirement already satisfied: numpy!=1.24.0,>=1.17 in ./.local/lib
/python3.10/site-packages (from seaborn) (1.24.1)
Requirement already satisfied: pillow>=6.2.0 in /usr/lib/python3/di
st-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (9.0.1)
Requirement already satisfied: packaging>=20.0 in ./.local/lib/pyth
on3.10/site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (23.0)
Requirement already satisfied: cycler>=0.10 in ./.local/lib/python
3.10/site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (0.11.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/lib/python3
/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (2.4.7)
Requirement already satisfied: fonttools>=4.22.0 in ./.local/lib/py
thon3.10/site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (4.3
9.4)
Requirement already satisfied: contourpy>=1.0.1 in ./.local/lib/pyt
hon3.10/site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (1.0.
7)
Requirement already satisfied: python-dateutil>=2.7 in ./.local/lib
/python3.10/site-packages (from matplotlib!=3.6.1,>=3.1->seaborn)
(2.8.2)
Requirement already satisfied: kiwisolver>=1.0.1 in ./.local/lib/py
thon3.10/site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (1.
4.4)
Requirement already satisfied: pytz>=2020.1 in /usr/lib/python3/dis
t-packages (from pandas>=0.25->seaborn) (2022.1)
Requirement already satisfied: tzdata>=2022.1 in ./.local/lib/pytho
n3.10/site-packages (from pandas>=0.25->seaborn) (2023.3)
Requirement already satisfied: six>=1.5 in /usr/lib/python3/dist-pa
ckages (from python-dateutil>=2.7->matplotlib!=3.6.1,>=3.1->seabor
n) (1.16.0)
Installing collected packages: seaborn
Successfully installed seaborn-0.12.2

[notice] A new release of pip available: 22.3.1 -> 23.1.2
[notice] To update, run: python3 -m pip install --upgrade pip
```

In [12]:
```
import csv
a=[]
print("\n The given training dataset")
with open('Enjoysports.csv','r')as csvfile:
    reader = csv.reader(csvfile)
    for row in reader:
        a.append(row)
        print(row)
num_attributes = len(a[0])-1
```

```
 The given training dataset
['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes']
['sunny', 'warm', 'high', 'strong', 'warm', 'same', 'yes']
['rainy', 'cold', 'high', 'strong', 'warm', 'change', 'no']
['sunny', 'warm', 'high', 'strong', 'cool', 'change', 'yes']
```

In [13]:
```
print("\n The intial value of hypothesis:")
S=['0']*num_attributes
G=['?']*num_attributes
print("\n The most specific hypothesis SO:[0,0,0,0,0,0]")
print("\n The most general hypothesis GO : [?,?,?,?,?,?]")
```

```
 The intial value of hypothesis:

 The most specific hypothesis SO:[0,0,0,0,0,0]

 The most general hypothesis GO : [?,?,?,?,?,?]
```

In [14]:
```
for j in range(0,num_attributes):
    S[j]=a[0][j];
```

In [15]:
```
S
```

Out[15]:
```
['sunny', 'warm', 'normal', 'strong', 'warm', 'same']
```

In [16]:
```
print("\n Candidate Elimation algorithm Hpothesis Version space compu
temp=[]
for i in range(0,len(a)):
    if a[i][num_attributes]=='yes':
        for j in range(0,num_attributes):
            if a[i][j]!=S[j]:
                S[j]='?'
        for j in range(0,num_attributes):
            for k in range(1,len(temp)):
                if temp[k][j]!='?'and temp[k][j]!=S[j]:
                    del temp[k]
        print("-----------------------------------------------------------
        print("For training Example No:{0} the hypothesis is S{0}".fo
        if(len(temp)==0):
            print("For training Example No:{0} the hypothesis is G{0}
        else:
            print("For Positive training Example No:{0} the hypothesi
    if a[i][num_attributes]=='no':
        for j in range(0,num_attributes):
            if S[j]!=a[i][j] and S[j]!='?':
                G[j]=S[j]
                temp.append(G)
                G=['?']*num_attributes
        print("-----------------------------------------------------------
        print("For training Example No:{0} the hypothesis is S{0}".fo
        print("For Positive training Example No:{0} the hypothesis is
```

```
 Candidate Elimation algorithm Hpothesis Version space computation

-----------------------------------------------------------------
For training Example No:1 the hypothesis is S1 ['sunny', 'warm', 'n
ormal', 'strong', 'warm', 'same']
For training Example No:1 the hypothesis is G1 ['?', '?', '?', '?',
'?', '?']
-----------------------------------------------------------------
For training Example No:2 the hypothesis is S2 ['sunny', 'warm',
'?', 'strong', 'warm', 'same']
For training Example No:2 the hypothesis is G2 ['?', '?', '?', '?',
'?', '?']
-----------------------------------------------------------------
For training Example No:3 the hypothesis is S3 ['sunny', 'warm',
'?', 'strong', 'warm', 'same']
For Positive training Example No:3 the hypothesis is G3 [['sunny',
'?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?',
'?', '?', '?', '?', 'same']]
-----------------------------------------------------------------
For training Example No:4 the hypothesis is S4 ['sunny', 'warm',
'?', 'strong', '?', '?']
For Positive training Example No:4 the hypothesis is G4 [['sunny',
'?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]
```

In [10]:
```
import math
import csv
def load_csv(filename):
    lines=csv.reader(open(filename,"r"))
    dataset=list(lines)
    headers=dataset.pop(0)
    return dataset,headers
class Node:
    def __init__(self,attribute):
        self.attribute=attribute
        self.children=[]
        self.answer=""
def subtables(data,col,delete):
    dic={}
    coldata=[row[col]for row in data]
    attr=list(set(coldata))
    counts=[0]*len(attr)
    r=len(data)
    c=len(data[0])
    for x in range(len(attr)):
        for y in range(r):
            if data[y][col]==attr[x]:
                counts[x]+=1
    for x in range(len(attr)):
        dic[attr[x]]=[[0 for i in range(c)] for j in range(counts[x])
        pos=0
        for y in range(r):
            if data[y][col]==attr[x]:
                if delete:
                    del data[y][col]   #removing tat particular colu
                dic[attr[x]][pos]=data[y] #all rows for each unique
                pos+=1
    return attr,dic
def entropy(S):
    attr=list(set(S))  #S will basically have last column data(not ne
    if len(attr)==1:
        return 0    #if there is either only yes/ only no =>entrop is
    counts=[0,0]
    for i in range(2):
        counts[i]=sum([1 for x in S if attr[i]==x])/(len(S)*1.0) #fi
    sums=0
    for cnt in counts:
        sums+=-1*cnt*math.log(cnt,2)  #base 2(second parameter)
    return sums
def compute_gain(data,col): #col  is column-header
    attr,dic = subtables(data,col,delete=False) #here no deletion, we
    total_size=len(data)  # |S| value in formula
    entropies=[0]*len(attr) #entropies of each value
    ratio=[0]*len(attr)   # to maintain |Sv|/|S| values
    total_entropy=entropy([row[-1] for row in data])

    for x in range(len(attr)):
        ratio[x]=len(dic[attr[x]])/(total_size*1.0) #len of dic=> |S
        entropies[x]=entropy([row[-1] for row in dic[attr[x]]])

        total_entropy-=ratio[x]*entropies[x] #acc to formula
    return total_entropy
def build_tree(data,features):
    lastcol=[row[-1] for row in data]
```

```
    if(len(set(lastcol)))==1:  #if last column contains either only
        node=Node("")          #we are not building the tree further
        node.answer=lastcol[0] #it'll be either yes/no
        return node
    n=len(data[0])-1   #-1 boz we dont need the last column values
    gains=[0]*n        # gain is initialized to be 0 for all attribu
    for col in range(n):
        gains[col]=compute_gain(data,col)  #compute gain of each att

    split=gains.index(max(gains))          # split will have the inde
    node=Node(features[split])             # features list will have
                                           # so now we create a subt
    fea = features[:split]+features[split+1:]
    attr,dic=subtables(data,split,delete=True)  #attr will have poss
                                                #dic will have all r
    for x in range(len(attr)):
        child=build_tree(dic[attr[x]],fea)     #for each value of th
        node.children.append((attr[x],child))  #again build the tree
    return node
def print_tree(node,level):
    if node.answer!="":              #if its a leaf node
        print(" "*level,node.answer) #just print "level" no of space
        return
    print(" "*level,node.attribute)  #attribute in the node
    for value,n in node.children:
        print(" "*(level+1),value)
        print_tree(n,level+2)        # recursive call to the next no

def classify(node,x_test,features): #features: column headers
    if node.answer!="":      #this will be true only for leaf nodes(
        print(node.answer)
        return
    pos=features.index(node.attribute) #node.attribute will have the
    for value, n in node.children:   #for every value of that attrib
        if x_test[pos]==value:       # for that particular value go
            classify(n,x_test,features) #go deeper in the tree
dataset,features=load_csv("traintennis.csv")
#lastcol=[row[-1] for row in dataset]
node1=build_tree(dataset,features)
print("The decision tree for the dataset using ID3 algorithm is")
print_tree(node1,0)

testdata,features=load_csv("testtennis.csv")
for xtest in testdata:   #xtest is each row in testdata
    print("The test instance:",xtest)
    print("The label for test instance:",end=" ")
    classify(node1,xtest,features)
```

```
The decision tree for the dataset using ID3 algorithm is
 Outlook
  Sunny
   Humidity
    High
     No
    Normal
     Yes
  Overcast
   Yes
  Rain
   Wind
```

```
    Strong
     No
    Weak
     Yes
The test instance: ['Overcast', 'Hot', 'Normal', 'Weak']
The label for test instance: Yes
The test instance: ['Sunny', 'Cool', 'High', 'Strong']
The label for test instance: No
The test instance: ['Overcast', 'Hot', 'High', 'Weak']
The label for test instance: Yes
The test instance: ['Rain', 'Mild', 'High', 'Strong']
The label for test instance: No
The test instance: ['Rain', 'Cool', 'Normal', 'Weak']
The label for test instance: Yes
```

In [4]:
```
import numpy as np
X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)
y = np.array(([92], [86], [89]), dtype=float)
X = X/np.amax(X,axis=0) # maximum of X array longitudinally
y = y/100

#Sigmoid Function
def sigmoid (x):
    return 1/(1 + np.exp(-x))

#Derivative of Sigmoid Function
def derivatives_sigmoid(x):
    return x * (1 - x)

#Variable initialization
epoch=5000  #Setting training iterations
lr=0.1      #Setting learning rate
inputlayer_neurons = 2      #number of features in data set
hiddenlayer_neurons = 3     #number of hidden layers neurons
output_neurons = 1      #number of neurons at output layer

#weight and bias initialization
wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bout=np.random.uniform(size=(1,output_neurons))

for i in range(epoch):
#Forward Propogation
    hinp=np.dot(X,wh)+ bh
    hlayer_act = sigmoid(hinp)      #HIDDEN LAYER ACTIVATION FUNCTIO
    outinp=np.dot(hlayer_act,wout)+ bout
    output = sigmoid(outinp)

    outgrad = derivatives_sigmoid(output)
    hiddengrad = derivatives_sigmoid(hlayer_act)

    EO = y-output                     #ERROR AT OUTPUT LAYER
    d_output = EO* outgrad

    EH = d_output.dot(wout.T)       #ERROR AT HIDDEN LAYER (TRANSPOS
    d_hiddenlayer = EH * hiddengrad

    wout += hlayer_act.T.dot(d_output) *lr      #REMEMBER WOUT IS 3*
    wh += X.T.dot(d_hiddenlayer) *lr

print("Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n" ,output)
```

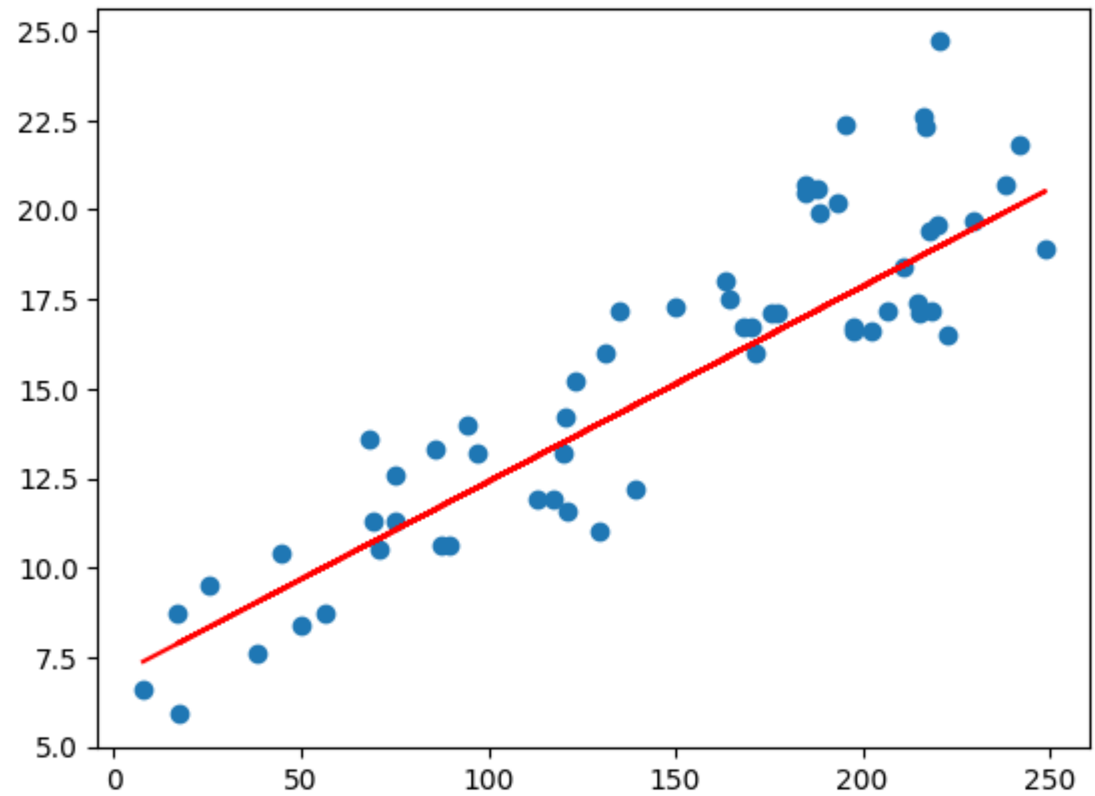
```
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
```

```
[[0.89483301]
[0.87666731]
[0.898617  ]]
```

In [17]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

#Reading the dataset
dataset = pd.read_csv("tv.csv")

#Setting the value for X and Y
x = dataset[['TV']]
y = dataset['Sales']

#Splitting the dataset
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size =

#Fitting the Linear Regression model
from sklearn.linear_model import LinearRegression
slr = LinearRegression()
slr.fit(x_train, y_train)

#Intercept and Coefficient
print("Intercept: ", slr.intercept_)
print("Coefficient: ", slr.coef_)


## Regression Equation: Sales = 6.948 + 0.054 * TV

#Prediction of test set
y_pred_slr= slr.predict(x_test)
#Predicted values
print("Prediction for test set: {}".format(y_pred_slr))

#Actual value and the predicted value
slr_diff = pd.DataFrame({'Actual value': y_test, 'Predicted value': y
slr_diff.head()

#Line of best fit
plt.scatter(x_test,y_test)
plt.plot(x_test, y_pred_slr, 'Red')
plt.show()

#Model Evaluation
from sklearn import metrics
meanAbErr = metrics.mean_absolute_error(y_test, y_pred_slr)
meanSqErr = metrics.mean_squared_error(y_test, y_pred_slr)
rootMeanSqErr = np.sqrt(metrics.mean_squared_error(y_test, y_pred_sl
print('R squared: {:.2f}'.format(slr.score(x,y)*100))
print('Mean Absolute Error:', meanAbErr)
print('Mean Square Error:', meanSqErr)
print('Root Mean Square Error:', rootMeanSqErr)
```

```
Intercept:  6.948683200001357
Coefficient:  [0.05454575]
Prediction for test set: [ 7.37414007 19.94148154 14.32326899 18.82
329361 20.13239168 18.2287449
 14.54145201 17.72692398 18.75238413 18.77420243 13.34144544 19.466
93349
```

```
 10.01415451 17.1923756  11.70507285 12.08689312 15.11418241 16.232
37035
 15.8669138  13.1068987  18.65965635 14.00690363 17.60692332 16.603
28147
 17.03419291 18.96511257 18.93783969 11.05597839 17.03419291 13.663
26538
 10.6796127  10.71234015 13.5487193  17.22510305  9.67597085 13.521
44643
 12.25053038 16.13418799 19.07965865 17.48692266 18.69783838 16.532
37199
 15.92145955 18.86693021 13.5050827  11.84143724  7.87050642 20.519
66653
 10.79961336  9.03233096 17.99419817 16.29237067 11.04506924 14.099
63141
 18.44147334  9.3759692   7.88687015  8.34505447 17.72692398 11.623
```




```
R squared: 81.10
Mean Absolute Error: 1.6480589869746525
Mean Square Error: 4.077556371826948
Root Mean Square Error: 2.019296008966231
```

In [6]:
```
import csv
import random
import math

def loadcsv(diabetes):
    dataset = list(csv.reader(open(filename,"r")))
    for i in range(len(dataset)):
        dataset[i] = [float(x) for x in dataset[i]]
    return dataset

def splitDataset(dataset, splitRatio):
    trainSize = int(len(dataset) * splitRatio)
    trainSet = []
    trainSet,testSet = dataset[:trainSize],dataset[trainSize:]
    return [trainSet, testSet]

def mean(numbers):
    return sum(numbers)/(len(numbers))

def stdev(numbers):
    avg = mean(numbers)
    v = 0
    for x in numbers:
        v += (x-avg)**2
    return math.sqrt(v/(len(numbers)-1))

def summarizeByClass(dataset):
    separated = {}
    for i in range(len(dataset)):
        vector = dataset[i]
        if (vector[-1] not in separated):
            separated[vector[-1]] = []
        separated[vector[-1]].append(vector)
    summaries = {}
    for classValue, instances in separated.items():
        summaries[classValue] = [(mean(attribute), stdev(attribute))
    return summaries

def calculateProbability(x, mean, stdev):
    exponent = math.exp((-(x-mean)**2)/(2*(stdev**2)))
    return (1 / math.sqrt(2*math.pi*(stdev**2))) * exponent

def predict(summaries, inputVector):
    probabilities = {}
    for classValue, classSummaries in summaries.items():
        probabilities[classValue] = 1
        for i in range(len(classSummaries)):
            mean, stdev = classSummaries[i]
            x = inputVector[i]
            probabilities[classValue] *= calculateProbability(x, mear
    bestLabel, bestProb = None, -1
    for classValue, probability in probabilities.items():
        if bestLabel is None or probability > bestProb:
            bestProb = probability
            bestLabel = classValue
    return bestLabel

def getPredictions(summaries, testSet):
    predictions = []
```

```
    for i in range(len(testSet)):
        result = predict(summaries, testSet[i])
        predictions.append(result)
    return predictions

def getAccuracy(testSet, predictions):
    correct = 0
    for i in range(len(testSet)):
        if testSet[i][-1] == predictions[i]:
            correct += 1
    return (correct/(len(testSet))) * 100.0

filename = '/home/nmit/Downloads/diabetes.csv'
splitRatio = 0.9
dataset = loadcsv(filename)
actual = []
trainingSet, testSet = splitDataset(dataset, splitRatio)
for i in range(len(testSet)):
    vector = testSet[i]
    actual.append(vector[-1])
print('Split {0} rows into train={1} and test={2} rows'.format(len(da
summaries = summarizeByClass(trainingSet) #will have (mean,sd) for a
predictions = getPredictions(summaries, testSet)
print('\nActual values:\n',actual)
print("\nPredictions:\n",predictions)
accuracy = getAccuracy(testSet, predictions)
print("Accuracy",accuracy)
```

```
Split 768 rows into train=691 and test=77 rows

Actual values:
 [1.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0,
0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.
0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0,
1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 0.
0, 0.0, 1.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 1.0,
0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0]

Predictions:
 [1.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0,
0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.
0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1.
0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0,
0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0]
Accuracy 76.62337662337663
```

In [2]:
```
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
import numpy as np
from sklearn.model_selection import train_test_split

iris_dataset=load_iris()

#display the iris dataset
print("\n IRIS FEATURES \ TARGET NAMES: \n ", iris_dataset.target_nar
for i in range(len(iris_dataset.target_names)):
    print("\n[{0}]:[{1}]".format(i,iris_dataset.target_names[i]))

print("\n IRIS DATA :\n",iris_dataset["data"])

#split the data into training and testing data
X_train, X_test, y_train, y_test = train_test_split(iris_dataset["dat

print("\n Target :\n",iris_dataset["target"])
#print("\n")
#print(len(iris_dataset["target"]))
print("\n X TRAIN \n", X_train)
print("\n X TEST \n", X_test)
print("\n Y TRAIN \n", y_train)
print("\n Y TEST \n", y_test)

#train and fit the model
kn = KNeighborsClassifier(n_neighbors=5)
kn.fit(X_train, y_train)

#predicting from model
x_new = np.array([[5, 2.9, 1, 0.2]])
print("\n XNEW \n",x_new)
prediction = kn.predict(x_new)
print("\n Predicted target value: {}\n".format(prediction))
print("\n Predicted feature name: {}\n".format(iris_dataset["target_r

i=1
x= X_test[i]
x_new = np.array([x])
print("\n XNEW \n",x_new)

for i in range(len(X_test)):
  x = X_test[i]
  x_new = np.array([x])
  prediction = kn.predict(x_new)     #predict method returns label
  print("\n Actual : {0} {1}, Predicted :{2}{3}".format(y_test[i],ir:
print("\n TEST SCORE[ACCURACY]: {:.2f}\n".format(kn.score(X_test, y_t
```

```
 IRIS FEATURES \ TARGET NAMES:
  ['setosa' 'versicolor' 'virginica']

[0]:[setosa]

[1]:[versicolor]

[2]:[virginica]

 IRIS DATA :
```

```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
```

In [7]:
```
import pandas as pd
```

In [8]:
```
import numpy as np
```

In [9]:
```
import matplotlib.pyplot as plt
```

In [10]:
```
a=pd.read_csv('seattle-weather.csv')
```

In [11]:
```
a
```

Out[11]:

| | date | precipitation | temp_max | temp_min | wind | weather |
|---|---|---|---|---|---|---|
| **0** | 2012-01-01 | 0.0 | 12.8 | 5.0 | 4.7 | drizzle |
| **1** | 2012-01-02 | 10.9 | 10.6 | 2.8 | 4.5 | rain |
| **2** | 2012-01-03 | 0.8 | 11.7 | 7.2 | 2.3 | rain |
| **3** | 2012-01-04 | 20.3 | 12.2 | 5.6 | 4.7 | rain |
| **4** | 2012-01-05 | 1.3 | 8.9 | 2.8 | 6.1 | rain |
| **...** | ... | ... | ... | ... | ... | ... |
| **1456** | 2015-12-27 | 8.6 | 4.4 | 1.7 | 2.9 | rain |
| **1457** | 2015-12-28 | 1.5 | 5.0 | 1.7 | 1.3 | rain |
| **1458** | 2015-12-29 | 0.0 | 7.2 | 0.6 | 2.6 | fog |
| **1459** | 2015-12-30 | 0.0 | 5.6 | -1.0 | 3.4 | sun |
| **1460** | 2015-12-31 | 0.0 | 5.6 | -2.1 | 3.5 | sun |

1461 rows × 6 columns

In [12]:
```
a=a.drop(['date'],axis=1)
```

In [13]:
```
a
```

Out[13]:

| | precipitation | temp_max | temp_min | wind | weather |
|---|---|---|---|---|---|
| **0** | 0.0 | 12.8 | 5.0 | 4.7 | drizzle |
| **1** | 10.9 | 10.6 | 2.8 | 4.5 | rain |
| **2** | 0.8 | 11.7 | 7.2 | 2.3 | rain |
| **3** | 20.3 | 12.2 | 5.6 | 4.7 | rain |
| **4** | 1.3 | 8.9 | 2.8 | 6.1 | rain |
| **...** | ... | ... | ... | ... | ... |
| **1456** | 8.6 | 4.4 | 1.7 | 2.9 | rain |
| **1457** | 1.5 | 5.0 | 1.7 | 1.3 | rain |
| **1458** | 0.0 | 7.2 | 0.6 | 2.6 | fog |
| **1459** | 0.0 | 5.6 | -1.0 | 3.4 | sun |
| **1460** | 0.0 | 5.6 | -2.1 | 3.5 | sun |

1461 rows × 5 columns

In [14]:
```
from sklearn.preprocessing import LabelEncoder
```

In [15]:
```
a['weather'].value_counts()
```

Out[15]:
```
weather
rain       641
sun        640
fog        101
drizzle     53
snow        26
Name: count, dtype: int64
```

In [17]:
```
l=LabelEncoder()
```

In [18]:
```
a['w']=l.fit_transform(a['weather'])
```

In [19]:
```
a
```

Out[19]:

| | precipitation | temp_max | temp_min | wind | weather | w |
|---|---|---|---|---|---|---|
| **0** | 0.0 | 12.8 | 5.0 | 4.7 | drizzle | 0 |
| **1** | 10.9 | 10.6 | 2.8 | 4.5 | rain | 2 |
| **2** | 0.8 | 11.7 | 7.2 | 2.3 | rain | 2 |
| **3** | 20.3 | 12.2 | 5.6 | 4.7 | rain | 2 |
| **4** | 1.3 | 8.9 | 2.8 | 6.1 | rain | 2 |
| **...** | ... | ... | ... | ... | ... | ... |
| **1456** | 8.6 | 4.4 | 1.7 | 2.9 | rain | 2 |
| **1457** | 1.5 | 5.0 | 1.7 | 1.3 | rain | 2 |
| **1458** | 0.0 | 7.2 | 0.6 | 2.6 | fog | 1 |
| **1459** | 0.0 | 5.6 | -1.0 | 3.4 | sun | 4 |
| **1460** | 0.0 | 5.6 | -2.1 | 3.5 | sun | 4 |

1461 rows × 6 columns

In [20]:
```
a=a.drop(['weather'],axis=1)
```

In [21]:
```
a
```

Out[21]:

| | precipitation | temp_max | temp_min | wind | w |
|---|---|---|---|---|---|
| **0** | 0.0 | 12.8 | 5.0 | 4.7 | 0 |
| **1** | 10.9 | 10.6 | 2.8 | 4.5 | 2 |
| **2** | 0.8 | 11.7 | 7.2 | 2.3 | 2 |
| **3** | 20.3 | 12.2 | 5.6 | 4.7 | 2 |
| **4** | 1.3 | 8.9 | 2.8 | 6.1 | 2 |
| **...** | ... | ... | ... | ... | ... |
| **1456** | 8.6 | 4.4 | 1.7 | 2.9 | 2 |
| **1457** | 1.5 | 5.0 | 1.7 | 1.3 | 2 |
| **1458** | 0.0 | 7.2 | 0.6 | 2.6 | 1 |
| **1459** | 0.0 | 5.6 | -1.0 | 3.4 | 4 |
| **1460** | 0.0 | 5.6 | -2.1 | 3.5 | 4 |

1461 rows × 5 columns

In [22]:
```
from sklearn.model_selection import train_test_split
```

In [23]:
```
a.columns
```

Out[23]:
```
Index(['precipitation', 'temp_max', 'temp_min', 'wind', 'w'], dtype
='object')
```

In [24]:
```
x=a[['precipitation','temp_max','temp_min','wind']]
y=a['w']
```

In [27]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.1,ranc
```

In [28]:
```
from sklearn.linear_model import LinearRegression
```

In [29]:
```
m=LinearRegression()
```

In [30]:
```
m.fit(x_train,y_train)
```

Out[30]:
```
▾ LinearRegression
LinearRegression()
```

In [31]:
```
m.predict([[8.6,4.4,1.7,1.3]])
```

```
/home/nmit/.local/lib/python3.10/site-packages/sklearn/base.py:439:
UserWarning: X does not have valid feature names, but LinearRegress
ion was fitted with feature names
  warnings.warn(
```

Out[31]:
```
array([1.82046527])
```

In [32]:
```
y_predict=m.predict(x_test).round(0)
```

In [33]:
```
y_test=y_test.round(0)
```

In [34]:
```
from sklearn.metrics import f1_score
```

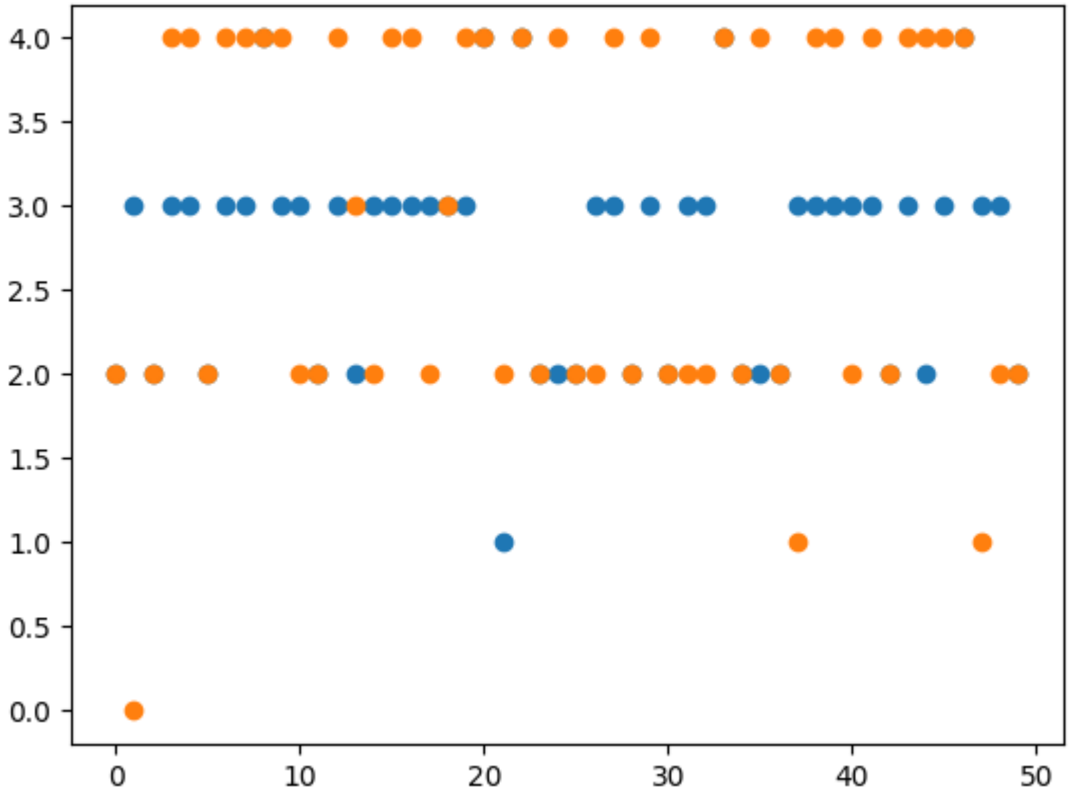
In [35]:
```
f1_score(y_test,y_predict,average='micro')
```

Out[35]:
```
0.35374149659863946
```

In [36]:
```
i=np.array(range(50))
```

In [37]:
```
plt.scatter(i,y_predict[0:50])
plt.scatter(i,y_test[0:50])
```

Out[37]: <matplotlib.collections.PathCollection at 0x7f091a39a500>




In [ ]: